CPSC 413 Fall 2010: Midterm November 2nd, 2010

Name: _____

ID#: _____

Instructions:

- No outside aids of any kind are allowed.
- You have 1 hour and 15 minutes.
- Do not turn this page until you are told to do so.
- There are four questions on this exam, and each question is worth a total of 10 points.

1. (a) *Define what it means for $f = O(g)$ (3 pts).*

   $f = O(g)$ if there is a natural number $m \geq 0$ and a constant $c > 0$ so that for all $n \geq m$, $f(n) \leq cg(n)$.

   (b) *Show that $ln(n^4) = O((ln(n))^2)$ (3 pts).*

   Indeed,

$$\lim_{n\to\infty} \frac{ln(n^4)}{(ln(n))^2} = \lim_{n\to\infty} \frac{4ln(n)}{(ln(n))^2} = \lim_{n\to\infty} \frac{4}{ln(n)} = 0$$

   so $ln(n^4) = O((ln(n))^2)$.

   (c) *Suppose that we have two non-negative functions f, g such that for $n \geq 100$, $f(n) \leq g(n)$. Show that $f + g = O(g)$ (4 pts).*

   For $n \geq 100$, $f(n) + g(n) \leq g(n) + g(n) = 2g(n)$, so $f + g = O(g)$.

2. Meghan is planning a cross-country trip along the Trans-Canada highway from Halifax to Vancouver. She knows the location of n gas stations along the trip (not neccesarily in order). Her car can only run k kilometers before she must stop for gas.

   (a) *Give an algorithm that can determine the least number of gas stations she must stop at along her journey so that she never runs out of gas, and show that your algorithm runs in $O(n \log n)$ time (4 pts).*

   The idea is to take the farthest gas station until we would run out of gas, each time. That is, the first station we pick is the highest g_i for which $g_i < k$, then find the next one less than $g_i + k$, and so on. Here is the pseudo-code for this algorithm:

   - sort the gas stations in increasing order of how far they are from halifax, giving a set $g_1, g_2, \ldots g_n$;
   - set $i = 1$;
   - set $c = k$;
   - initialize an empty list l;
   - while $c \leq$ (distance from halifax to vancouver);
     - if $g_i > c$, add $i - 1$ to the list l and set $c = g_{i-1} + c$;
     - increase i;
   - endwhile;
   - return l.

   (b) *Prove that your algorithm always return the minimum number of gas stations (6 pts).*

   We will use a "stays-ahead" argument. Let a_i be the gas stations chosen by our algorithm. Suppose S is any other solution, which chooses gas stations at locations $s_1, s_2, \ldots s_k$. Assume the s_i are sorted by their distance from Halifax.

   We claim that for any i, $s_i \leq a_i$. We prove this by induction on n. For $n = 1$, since S is a solution, $s_1 \leq k$. But we chose a_1 so that a_1 was the furthest gas station less than k, so $s_1 \leq a_1$. Now suppose the result is true for n, so $s_n \leq a_n$. Since S is a solution, it must choose a gas station k kilometers after s_n, so $s_{n+1} \leq s_n + k$. But

$s_n \leq a_n$, so $s_{n+1} \leq a_n + k$. But a_{n+1} was chosen so that it was the farthest gas station so that $a_{n+1} \leq a_n + k$. Thus $s_{n+1} \leq a_{n+1}$ as required.

We now prove that our algorithm is optimal. Since that O is an optimal solution, and O uses fewer gas stations than A, say m of them. Assume O's gas stations are sorted by distance from Halifax. By our previous claim, $o_m \leq a_m$. But our algorithm chooses at least one more gas station a_{m+1}, so $a_m + k$ is less than the distance to Vancounver. Thus, $o_m + k$ is less than the distance to Vancouver, so O cannot be a solution: contradiction. Thus O cannot have fewer gas stations than our algorithm, so our algorithm is optimal.

3. *Suppose that a divide and conquer algorithm splits an input of size n into 3 pieces, each of size $\frac{n}{3}$, while the time taken to split and recombine the data is $O(n)$. That is, if $T(n)$ is the running time of the algorithm on n inputs, then there is some constant c so that*

$$T(n) \leq 3T(n/3) + cn \text{ and } T(3) \leq c.$$

Show that the running time of the algorithm is $O(n \log n)$ (10 pts).

We analyze the running time of the algorithm at each "level" of the recursion. At the initial level, the total running time without the recursive calls is cn. At the first level of the recursion, there are three subproblems of size $\frac{n}{3}$ each, so the total is $c\frac{n}{3} + c\frac{n}{3} + c\frac{n}{3} = cn$. At the second level of the recursion, there are 9 subproblems of size $\frac{n}{9}$ each, so the total is again cn.

It takes $\log_3 n$ levels of the recursion to reach the base case, so our total running time is

$$\sum_{i=0}^{\log_3 n - 1} cn = cn \log_3 n = O(n \log n),$$

as required.

4. *Given an array $[a_1, a_2, \ldots a_n]$ of distinct integers, say that a_i is a* local minimum *if a_i is strictly less than both of its neighbours (an endpoint is considered a local minimum if it is strictly less than its single neighbour).*

   (a) *Give an algorithm for finding a local minimum while only looking at $O(\log n)$ elements of the array (5 pts).*

   The idea is to look at the middle element of the array. If it is a local minimum, we are done. If it is not, there is an array element on one side or the other that is higher. We recursively run the algorithm on that half of the array. In pseudo-code:

   - FindLocalMin(input: array A)
   - if $|A| = 1$, return a_1,
   - else
     - let a_i be the middle element of the array,
     - if a_i is a local minimum, return a_i,
     - else if $a_{i-1} < a_i$, run FindLocalMin on $a_1, a_2 \ldots a_{i-1}$,
     - else run FindLocalMin on $a_{i+1}, a_{i+2}, \ldots a_n$.
     - endif
   - endif

   At each stage, the algorithm splits into a single recursive call of size $\frac{n}{2}$, and only spends constant time outside of the recursion. At most, it takes $\log_2 n$ recursive steps to reach the base case, so the total running time is $O(\log n)$.

   (b) *Prove that your algorithm does find a local minimum (5 pts).*

   We need to check that an array element returned from a recursive call is a local minimum for the entire array. If the element returned checked both of its neighbours, then it is a local minimum for the entire array. On the other hand, suppose the element v that was returned did not check one of its neighbours w in the whole array. However, by the construction of our recursive calls, $v > w$, so v is larger than this neighbour. Thus v would be a local minimum for the entire array.